

INTRODUCCION A LARAVEL 4.2

ANDRES FELIPE SOTO RAMIREZ

JUAN DAVID RAMOS TELLO

GESTION DE CARTAS LABORALES – GCL

SANTIAGO DE CALI

2015



Table de Contenido



1. Filosofía Laravel

Laravel es un framework para aplicaciones web con sintaxis expresiva y elegante. Creemos que el desarrollo debe ser una experiencia agradable y creativa para que sea verdaderamente enriquecedora. Laravel busca eliminar el sufrimiento del desarrollo facilitando las tareas comunes utilizadas en la mayoría de los proyectos web, como la autenticación, enrutamiento, sesiones y almacenamiento en caché.

Laravel pretende hacer que el proceso de desarrollo sea agradable para el desarrollador sin sacrificar funcionalidades de la aplicación. Desarrolladores felices hacen mejor código. Con este fin, hemos intentando combinar lo mejor que hemos visto en otros frameworks web, incluyendo frameworks de otros lenguajes, como Ruby on Rails, ASP.NET MVC y Sinatra.

Laravel es accesible, pero potente, ofreciendo herramientas poderosas necesarias para aplicaciones de gran envergadura. Un espléndido contenedor de inversión de control, sistema de migraciones expresivo, y un soporte para pruebas unitarias estrechamente integrado, te entregan las herramientas que necesitas para construir cualquier aplicación que te hayan encargado.

Laravel es un framework MVC, desarrollado a base de los frameworks Zend y symfony.

Todo lo que tiene que ver con estilos CSS, archivos JavaScript, plantillas, plugins etc. se encuentra alojados en la carpeta public.

Aquí otra explicación del framework: http://prezi.com/gejdolwh91pz/?utm_campaign=share&utm_medium=copy&rc=ex0share



2. Enrutamiento básico

Primero que todo debemos comprender que lo primero que se hace para que podamos ver alguna vista va a ser escribir una la ruta.

¿Qué hace una ruta? Verán, una ruta recibe una petición del usuario y luego de un determinado proceso debe devolver una respuesta, toda las rutas deben comenzar con la palabra reservada Route y dependiendo si se envía algún formulario o no, se escribe Route::get o si se envía algún formulario Route::post.

La mayoría de las rutas de la aplicación serán definidas en el archivo app/routes.php. Las rutas más simples en Laravel consisten en una URI y la ejecución de un Closure (función anónima).

Entonces esto:

```
Route::get('/', function()
{
    return View::make('hello');
});
```

Significa que cuando el usuario solicite (GET), la página de inicio (/) enviara a la vista hello (views/hello.php).

Esta es la forma más sencilla para enviar a una vista:

```
Route::get('/',function(){
    return View::make('usuarios.usuarios', array('usuario'=> $usuario));
});
//---------->
```

View::make() recibe como primer parámetro la vista donde se envía, en primer lugar va el nombre de la carpeta donde se encuentra el archivo (Views/carpeta/vista), separado por un punto y después el nombre del archivo. Como segundo parámetro recibe un arreglo con datos, este es opcional.



Nuestras rutas pueden tener "parámetros" que nos permitirán tener rutas dinámicas, y más complejas, por ejemplo:

```
Route::get('hello/{usuario}', function($usuario)
{
    return "Hello $usuario";
});
```

Así pueden tener una URL dinámica que responda a cualquier nombre, se encierra la variable en unos {}, nada complejo de entender. Lo mismo nos serviría para enviar algún otro valor como un id.

Asociar rutas a controladores y acciones:

```
//Ruta para crear un nuevo usuario
Route::get('home/usuarios/crear', 'usuarioController@crearUsuario');
```

Cuando la URL sea home/usuarios/crear**usaremos el controlador llamado usuarioController y la función crearUsuario.**

Cuando se ENVIA algún formulario y quieres realizar alguna acción en un controlador:

```
//Ruta cuando se envie el formulario con los datos del nuevo usuario.
Route::post('home/usuarios/crear', 'usuarioController@agregarUsuario');
```

Cuando la URL sea home/usuarios/crear **PERO se ha enviado un formulario, lo podemos deducir al ver Route::post, invocaremos al controlador usuarioController y se usara la función llamada agregarUsuario.**



3. Entradas de datos y validaciones

Puedes acceder a los datos enviados de un formulario de la siguiente manera:

```
$input = Input::all();
```

VALIDACIONES:

En el proyecto GCL se uso el siguiente algoritmo desde un modelo para después llamarlo desde el controlador. Explicare a continuación el paso a paso.

1. Esta es una función que debe ser public y static para poder usarla como la usamos nosotros en el controlador

```
//Agregar usuario
public static function agregarUsuario($input){
    // función que recibe como parámetro la inf
```

2. Se declara un arreglo con reglas de validación para cada valor:

```
// Declaramos reglas para validar
$reglas =  array(

    'numIdentificacion' => array('required', 'numeric', 'max:99999999999', 'min:999999',
    'nombres'  => array('required','alpha', 'max:100', 'min:2'),
    'apellidos' => array('required','alpha', 'max:100', 'min:2'),
    'direccion' => array( 'min:5'),
    'fechanac' => array('required'),
    'telefono' => array('max:10000000000', 'min:100000', 'numeric'),
    'sexo' => array('required'),
    'correo' => array('required','max:100', 'min:4', 'email'),
    'password' => array('required', 'max:100', 'min:5', 'same:password2'),
    'password2' => array('required','same:password', 'max:100', 'min:5'),
    'idrol' => array('required'),
);
```



3. El paso a seguir es crear un objeto de la clase Validator de laravel e invocar la función make y darle como parámetro la variable que almaceno los valores enviados del formulario y las reglas de validación:

```
$validator = Validator::make($input, $reglas);
```

4. Después usamos un condicional:

```
// verificamos que los datos cumplan la validación
if ($validator->fails()){

    // si no cumple las reglas se van a devolver los errores al
    $respuesta['mensaje'] = $validator;
    $respuesta['error']   = true;
}else{

    //En caso de cumplir con la validacion
    //se encripta la contraseña
    $input['password']=Hash::make($input['password']);
    $input['nombres']=strtoupper($input['nombres']);
    $input['apellidos']=strtoupper($input['apellidos']);

    //se crea el objeto usuario
    $usuario = static::create($input);

    // se retorna un mensaje de éxito al controlador
    $respuesta['mensaje'] = '¡Usuario creado con exito!';
    $respuesta['error']   = false;
    $respuesta['data']    = $usuario;
}

return $respuesta;

}    //fIN AGREGAR USUARIO
```

- Si el objeto $validator que creamos anteriormente tiene error en las reglas de validación, en una variable $respuesta, que es un arreglo vacío, le asignamos el valor al subíndice [‘error’] que sea ‘true’ y en el subíndice [‘mensaje’] los mensajes de validación del framework (Cabe resaltar que los mensajes del framework son en inglés, así que se crea



una carpeta en app/lang con los mensajes en español, para que el framework utilice estos mensajes y no los que vienen por defecto. Se configura en app/config/app.php y buscamos la línea ‘locale’ =>’en’ y se cambia por el nombre de la carpeta creada en app/lang, en este caso ‘locale’ =>’es’)

- Si por el contrario, no hubo errores en la validación encriptamos la contraseña y usamos una función de php para convertir a mayúscula los caracteres y creamos el usuario. Por ultimo en el arreglo que esta vacío llamado $respuesta en el subíndice [‘mensaje’] le asignamos un mensaje de éxito al crear el usuario y en el subíndice [‘error’] le damos un valor de ‘false’

Por ultimo retornamos la variable $respuesta que usaremos a continuación en el controlador

5. Ya en el controlador:

```
//Funcion para agregar un nuevo usuario
public function agregarUsuario(){

    $input = Input::all();
    //Del modelo Usuario usamos la funcion agregarUsuario que recibe como parametro
    //los datos enviados
    $respuesta = Usuario::agregarUsuario($input);

        if ($respuesta['error'] == true){
            return Redirect::to('home/usuarios/crear')->withErrors($respuesta['mensaje'] )->withInput();
        }else{
            return Redirect::to('home/usuarios/crear')->with('mensaje', $respuesta['mensaje']);
        }
} //-->
```

- Primero capturamos en una variable $input el valor de los campos enviados del formulario
- Creamos una variable $respuesta que es un objeto de la clase Usuario y que utiliza la función agregarUsuario() que explicamos anteriormente. Recibe como parámetro una variable con el valor del formulario enviado
- SI la variable $respuesta[‘error’] tiene un valor de true envía los mensajes de error
- Si no hay error Envía un mensaje de éxito a la vista usuarios porque el usuario fue creado satisfactoriamente.



4. SubLengüaje BLADE

Blade es básicamente un sub-lenguaje muy sencillo, que antes de ser usado por nuestra aplicación, es compilado a PHP plano. Para usar Blade, simplemente creen sus plantillas en el directorio **views/** con la extensión **.blade.php** en vez de .php.



En nuestra vista si usaramos solo php escribiríamos una variable de la siguiente manera:

```
<h1>Hello <?php echo $name ?></h1>
```

Con blade:

```
<h1>Hello {{ $name }}</h1>
```

Básicamente: **{{ }}** es un sustituto de **<?php ?>** que es más corto y fácil de escribir.

USO DE LAYOUTS EN LARAVEL CON BLADE

Ese HTML que se repite en cada página se llama layout y en Blade podemos escribirlo así:

Creamos una vista llamada layout.blade.php en Views con el siguiente HTML:



```
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Aprendiendo Laravel</title>
  </head>
  <body>
      @yield('content')
      <hr />
      Copyright 2013 - Todos los ponies reservados.
  </body>
</html>
```

Vemos que yield(‘contenido’) será la parte variable del layout y lo utilizaremos de la siguiente forma:

En Views creamos el archivo template.blade.php con el siguiente código:

```
@extends ('layout')

@section ('content')

  <h1>Hello {{ $name }}</h1>

@stop
```

Como vemos se usa @extends(‘nombre del layout’) para invocar el layout que quieres cargar y @section(‘content) que será la pare donde escribiras el contenido de tu página.

CREAR FORMULARIOS CON BLADE:

Código:

```
{{ Form::open(array('url' => 'home/usuarios/crear', 'role' => 'form')) }}
```

```
  {{ Form::close() }}
```



Estos dos helpers (Form::open y Form::close) se encargan de abrir y cerrar las etiquetas HTML del formulario, respectivamente, pero no sólo eso:

Form::open:

- Genera URLs a través de la clase de Rutas de Laravel.
- Permite emular los métodos PUT y DELETE (dado que los navegadores sólo soportan GET y POST).
- Genera un token para protección anti ataques CSRF, lo cual está habilitado en Laravel 4 automáticamente. Más sobre ataques CSRF.
- Entre otros

Form::close:.

- Mantiene concordancia con Form::open (nuestro IDE probablemente protestaría si usamos </form> sin antes haber usado explícitamente <form*>

Hay que resaltar, sobre todo, en el array el primer atributo:

```
(array('url' => 'home/usuarios/crear'
```

Esta sería la ruta a la que enviaremos a nuestro archivo Routes.php. Anteriormente explique como recibir los datos cuando se envía un formulario.

Ahora vamos a definir los campos del formulario:

```
<div class="form-group">
  {{Form::label('numIdentificacion', 'Numero de identificación: *')}}
  {{Form::text('numIdentificacion', Input::old('numIdentificacion'), array('class'=>'form-control',
</div>
```

Form::label:

- Recibe como primer parámetro el atributo name de una etiqueta HTML y de segundo parámetro el valor que se imprime en pantalla.

Form::text:

- Es un input tipo texto
- Recibe como primer parámetro el atributo name de la etiqueta HTML INPUT



- Input::old se usa cuando a la hora de la validación exista un error devuelva el valor anterior que este tenia
- Array() recibe como parámetro las demás etiquetas HTML que quieras añadirle al input, tales como class, id, placeholder, etc.

Sigue este enlace para conocer más sobre los tipos de input de laravel:

http://laraveles.com/docs/4.2/html#text

Ahora veamos el botón para enviar y resetear:

```
{{Form::submit('Guardar', array('class'=>'btn btn-warning'))}}
{{Form::reset('Resetear', array('class'=>'btn btn-default'))}}
```

En primer parámetro se espera el nombre a imprimir en pantalla y de segundo parámetro demás atributos que quieras añadir al botón.



5. Uso de los Models

El ORM llamado Eloquent incluido en Laravel provee una hermosa y sencilla implementación de ActiveRecord (registro activo) para trabajar con tu base de datos. Cada tabla de la base de datos tiene un "Modelo" correspondiente, el cual es utilizado para interactuar con esa tabla.

Antes de comenzar, asegúrate de haber configurado la conexión a una base de datos en app/config/database.php.

DEFINIENDO UN MODELO:

Todos los modelos que crees en laravel 4.2 deberán extender de Eloquent y en una variable protected $table será el nombre de la tabla:

```
class User extends Eloquent {

    protected $table = 'usuarios';

}
```

En otra variable del mismo tipo, llamada $filiable irá el nombre de los atributos de la tabla:

```
protected $fillable = array('first_name', 'last_name', 'email');
```

Quedaría de esta manera:

```
Class Usuario extends Eloquent implements UserInterface{

    protected $table = 'usuarios';
    protected $fillable = array('numIdentificacion','nombres','apellidos','direccion'
```

Preferiblemente en esta parte debe estar lo relacionado con la BD, inserción de datos, consultas, modificaciones etc.



6. Uso básico de base de datos

La configuración de la base de datos se encuentra en la ruta app/config/database.php.

La conexión a la base de datos es extremadamente fácil:

```
'mysql' => array(
    'driver'    => 'mysql',
    'host'      => 'localhost',
    'database'  => 'pruebalaravel',
    'username'  => 'root',
    'password'  => 'CLAVE_ULTRA_SECRETA',
    'charset'   => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix'    => '',
),
```

Selecciona el gestor de base de datos, en este caso MySql, el host que en el ejemplo es localhost, el nombre de la base de datos, el username y su contraseña.

GENERADOR DE CONSULTAS:

Obtener todos los valores de una tabla

```
$users = DB::table('users')->get();
```

Realizar una consulta Where

```
$users = DB::table('users')->where('votes', '>', 100)->get();
```

Insertar un registro a una tabla




```
DB::table('users')->insert(
    array('email' => 'john@example.com', 'votes' => 0)
);
```

Insertar varios registros a una tabla

```
DB::table('users')->insert(array(
    array('email' => 'taylor@example.com', 'votes' => 0),
    array('email' => 'dayle@example.com', 'votes' => 0),
));
```

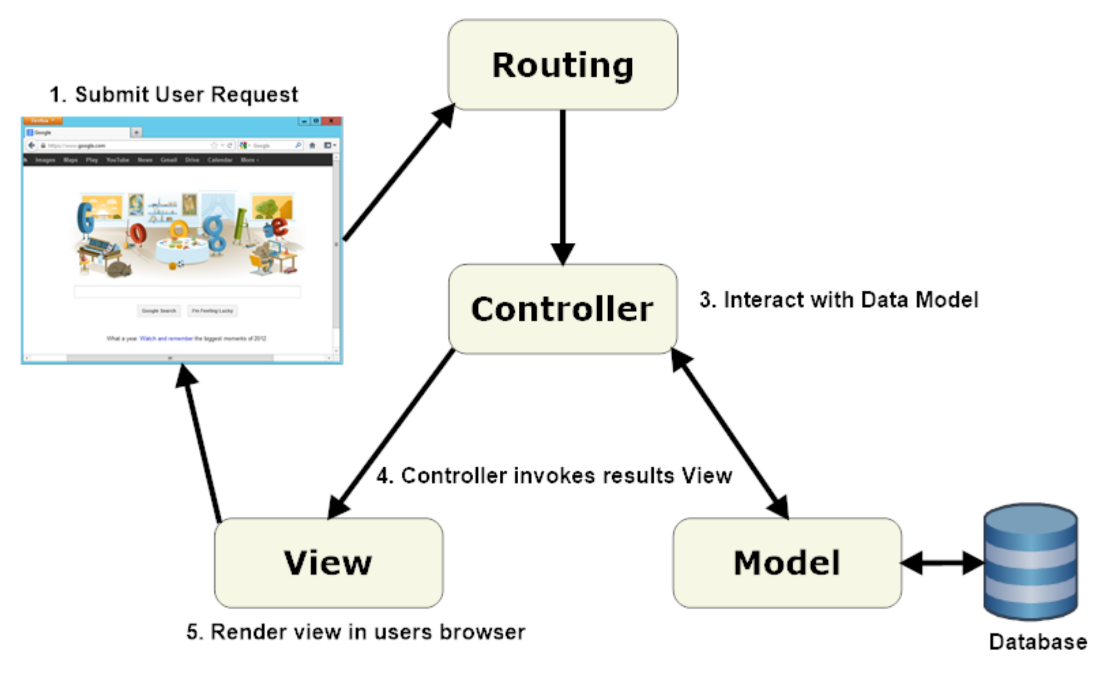
Actualizar registro de una tabla

```
DB::table('users')
            ->where('id', 1)
            ->update(array('votes' => 1));
```

Más información Sobre Eloquent ORM: http://laraveles.com/docs/4.2/queries



7. ESQUEMA DEL SISTEMA LARAVEL



8. DONDE BUSCAR INFORMACIÓN MAS DETALLADA ACERCA DE LARAVEL

http://laraveles.com/docs/4.2/queries

https://styde.net/category/aprende-laravel-desde-cero/

http://codehero.co/series/laravel-4-desde-cero.html